L Number	Hits	Search Text	DB	Time stamp
1	2	6134673.pn.	USPAT; EPO; JPO; DERWENT; IBM TDB	2001/05/31 12:09
7	0	6134673.pn. and preserv$3	USPAT; EPO; JPO; DERWENT; IBM TDB	2001/05/31 12:09
13	1	6182198.pn. and preserv$3	USPAT; EPO; JPO; DERWENT; IBM TDB	2001/05/31 12:10
19	0	5951695.pn. and preserv$3	USPAT; EPO; JPO; DERWENT; IBM TDB	2001/05/31 12:10
25	0	5968185.pn. and preserv$3	USPAT; EPO; JPO; DERWENT; IBM TDB	2001/05/31 12:11
31	0	5983360.pn. and preserv$3	USPAT; EPO; JPO; DERWENT; IBM TDB	2001/05/31 12:11
37	29546	(714/$).ccls.	USPAT; EPO; JPO; DERWENT; IBM TDB	2001/05/31 12:11
43	3	((714/$).ccls.) and preserv$3 same process same (failure or error$1) same (diagnos$3 or analy$4)	USPAT; EPO; JPO; DERWENT; IBM TDB	2001/05/31 12:17
49	3213	(714/25,1,2,6,9,11,13,15,25,33,37,742,759,798,805).ccls.	USPAT; EPO; JPO; DERWENT; IBM TDB	2001/05/31 12:22
55	8	((714/25,1,2,6,9,11,13,15,25,33,37,742,759,798,805).ccls. and (fail$3 or error) same (diagnos$ or analyz$3) same preserv$3	USPAT; EPO; JPO; DERWENT; IBM TDB	2001/05/31 12:32
61	1021	(714/47,48,49,4625).ccls.	USPAT; EPO; JPO; DERWENT; IBM TDB	2001/05/31 12:33
67	1	((714/47,48,49,4625).ccls.) and (fail$3 or error$1) same diagnos$4 same anal$	USPAT; EPO; JPO; DERWENT; IBM TDB	2001/05/31 12:35
68	52	((714/47,48,49,4625).ccls.) and diagnos$4 same analy$3	USPAT; EPO; JPO; DERWENT; IBM TDB	2001/05/31 12:41
74	1	"52" and preserv$3	USPAT; EPO; JPO; DERWENT; IBM TDB	2001/05/31 12:41
80	0	"52" and database	USPAT; EPO; JPO; DERWENT; IBM TDB	2001/05/31 12:42
86	12	(((714/47,48,49,4625).ccls.) and diagnos$4 same analy$3) and database	USPAT; EPO; JPO; DERWENT; IBM TDB	2001/05/31 12:42

L Number	Hits	Search Text	DB	Time stamp
1	2873	(714/25,26,27,31,33,34,41,42,48,6,8,9,11,12).CCLS.	USPAT; EPO; JPO; DERWENT; IBM TDB	2001/05/30 17:47
7	59	((714/25,26,27,31,33,34,41,42,48,6,8,9,11,12).CCLS.) and (fault or fail$3 or error$1) same (backup or back adj up) same (server or computer or workstation) same (database or application$1))	USPAT; EPO; JPO; DERWENT; IBM TDB	2001/05/30 19:03
13	14	((714/25,26,27,31,33,34,41,42,48,6,8,9,11,12).CCLS.) and (fault or fail$3 or error$1) same (backup or back adj up) same (server or computer or workstation) same (database or application$1)) and (freez or idle)	USPAT; EPO; JPO; DERWENT; IBM TDB	2001/05/30 19:05
19	14	((714/25,26,27,31,33,34,41,42,48,6,8,9,11,12).CCLS.) and (fault or fail$3 or error$1) same (backup or back adj up) same (server or computer or workstation) same (database or application$1)) and (freez$ or idle) same process$3	USPAT; EPO; JPO; DERWENT; IBM TDB	2001/05/30 19:06

L Number	Hits	Search Text	DB	Time stamp
1	0	dbms same (backup adj server) same identif$7 same (freez$ or idle or suspend$2) same process$3	USPAT	2001/05/30 14:37
2	0	dbms same (backup adj server) same identif$7 same (freez$ or idle or suspend$2)	USPAT	2001/05/30 14:38
3	0	dbms same backup same server same identif$7 same (freez$ or idle or suspend$2)	USPAT	2001/05/30 14:39
4	2	backup same server same identif$7 same (freez$ or idle or suspend$2)	USPAT	2001/05/30 14:46
5	5	(backup near5 server) same (freez$ or idle or suspend$2) same fail$3	USPAT	2001/05/30 14:52
6	4	(backup near5 server) same dbms	USPAT	2001/05/30 15:04
7	17417	(714/$).ccls.	USPAT	2001/05/30 15:04
8	204	((714/$).ccls.) and (fail$4 or fault$1) same (backup or back up) same server	USPAT	2001/05/30 15:05
9	13	(((714/$).ccls.) and (fail$4 or fault$1) same (backup or back up) same server) and dbms	USPAT	2001/05/30 15:19
10	5	((714/$).ccls.) and diagno$3 same failure same access$ same second same (server or computer)	USPAT	2001/05/30 15:24
11	430	(714/25).ccls.	USPAT	2001/05/30 15:24
12	1	((714/25).ccls.) and fail$3 same analy$3 same (backup or back adj up)	USPAT	2001/05/30 15:27
13	1	((714/25).ccls.) and fail$3 same analy$3 same access$3 same database	USPAT	2001/05/30 15:28

DOCUMENT-IDENTIFIER: US 5170480 A
TITLE: Concurrently applying redo records to backup database in a log sequence using single queue server per queue at a time

DEPR:
Once commutated to the queues 66, the REDO records are applied to the tracking database 72 by the queue servers. For example, the REDO records in queue b are obtained, one-by-one, by the queue server 68 and then used to update their respective pages. The sequence in which these changes occurred in the active system is preserved on the active transaction log 53, through the record transfer process to the tracking system 60, and on the active log data set 62. The records are hashed in their record log sequence onto the record queues 66. Consequently, the correct chronological update activity for each unit of transfer (page) is reflected by the REDO record sequence on the queue associated with the unit of transfer.

DEPR:
FIG. 4 illustrates the best mode of practicing the invention described above. In FIG. 4, a REDO record 105 is passed to a process QUEUE.sub.-- REDO.sub.-- RECORD 110. The process 110 passes the REDO record 105 to the appropriate queue, while preserving the original transaction log sequence to ensure a correct database. The process 110 includes a hashing procedure that divides the record block number (RBN) of the REDO record 105 by a number n in a data object 122 which is labelled NUMBER.sub.-- OF.sub.--

QUEUES. The index
obtained is used with a WORK.sub.-- QUEUE.sub.-- TABLE 123 to identify a
specific WORK.sub.-- QUEUE.sub.-- ENTRY 125. The indexed WORK.sub.--
QUEUE.sub.-- ENTRY 125 has pointers 127 and 129 to the last and first REDO
record 130 in the particular record queue headed by the WORK.sub.--
QUEUE.sub.-- ENTRY 125. The WORK.sub.-- QUEUE.sub.-- ENTRY 125 also includes a
SERVER.sub.-- PROCESS 128 pointing to a PROCESS.sub.-- INFO control block for a
queue server process 175. The queue server process 175 is awakened by setting
the RESUME.sub.-- STATUS flag 172 in the PROCESS.sub.-- INFO control block for
the server. It is asserted that each queue server has its own PROCESS.sub.--
INFO control block, and that each WORK.sub.-- QUEUE.sub.-- ENTRY identifies
only a single queue server by way of its SERVER.sub.-- PROCESS field.

CCXR:
714/16

CCXR:
714/20

DOCUMENT-IDENTIFIER: US 5828569 A
TITLE: Method and apparatus for maintaining network connections across a voluntary process switchover

BSPR:
This invention relates to a method and apparatus for maintaining network connections across a voluntary process switchover. A "takeover" or "switchover" is defined as a switch between processors in a dual processor environment, where one processor backs up the other so that in the event of a failure the backup processor takes over the responsibilities of the primary processor. In the past, network connections between applications that are coordinated through an application running in the primary processor have been lost during takeovers or switchovers. The present invention is directed to enhancing a smooth transition during takeovers, preferably during voluntary takeovers, so that no connections between server and client applications are lost.

DEPR:
One focus of the present invention is the checkpointing of certain data during a voluntary switch between primary and backup nodes of a protocol process node, also known more generally as a "takeover" or "switchover". A "takeover" or "switchover" is defined as a switch, either voluntary or involuntary, between processors such as found in primary and backup nodes 22, 24. In such a switchover the backup processor of the protocol process

processor takes over
the duties of the primary processor. Involuntary takeovers can occur in a
variety of ways, usually unexpected, such as if the primary processor is
damaged or data lines to the primary processor are corrupted. Voluntary
takeovers are "planned for" takeovers, and can occur by human intervention
(such as during load balancing) or automatically after some event (such as
after the failure of certain hardware ports). Furthermore, after a voluntary
switchover where the primary processor has not been damaged, and whenever
possible, the backup node becomes the primary node, and the primary node backs
up the backup node. The present invention allows for a smooth transition
during takeovers, preferably during voluntary takeovers, so that no connections
between server and client applications are lost. Performing a checkpoint will
increase the probability of maintaining a connection between a server
application and a client application in the event of a switchover.

DEPR:
In the present invention, the ACCEPT() function call need not be checkpointed
after it has been first called by the SPX server application. The absence of
checkpointing at this stage is indicated by the absence of arrows at reference
number 72 in FIG. 2., indicative of the lack of communication between the
primary and backup nodes at this time. Thus the server application
communicates with the primary processor that the ACCEPT() function has been
executed, as per arrow 70, but no checkpointing of data relating to the

ACCEPT() function call occurs at this point. In the present invention the checkpointing of the ACCEPT function call after it has been called is not needed because in the event of a failure of the primary processor at this point the server node would automatically redirect its requests to the backup processor, because it runs under the fault tolerant Nonstop kernel. The automatic redirection of requests to the backup processor is also described in the U.S. patent application entitled "Network System with Resilient Virtual Fault Tolerant Sessions" by Marc Desgrousilliers, commonly assigned, filed concurrently with the present invention and incorporated by reference herein.

DEPR:
However, in a preferred embodiment of the present invention, as represented by FIG. 4, even in the case of a such a voluntary switchover a connection is not in fact maintained if any data packets are being queued by the primary protocol process--which correspond to data packets waiting to be read by the SPX server, or data packets waiting to be transmitted to the client node application--that is, if the SPX server/ protocol process application connection is "non idle". Since in most instances a connection is "idle" anyway, for the most part there is no need to plan for "non idle" states. This is because typically no queue is present while a server application is reading, and the transmission of data packets between the protocol processor and client application is efficient enough that relatively little retransmission and waiting for acknowledgement of data occurs. However, from the above teaching one can

modify the FIG. 4
embodiment of practicing the present invention so that even queued data is
checkpointed just prior to a voluntary takeover, thus obviating the distinction
between "idle" and "non-idle" server/ protocol processor application
connections, and allowing connections between client and server applications to
always be maintained in a voluntary takeover, when in the SEND() and RECV()
states.

DEPR:
At step 250, the server application confirms that the server application is an
SPX server (sequential packet exchange server), and if not, there is no need to
proceed and the server application exits (step 252). If the server is a SPX
server, the application proceeds to the ACCEPT() function (step 255), which is
a function that puts an SPX server application in a state in which it awaits to
receive data from a client application. A takeover at this point would allow
the backup node to takeover the primary node functions, with the understanding
that there is no need to checkpoint an ACCEPT() function, because the NonStop
kernel that the protocol process node runs under would automatically redirect
requests directed to the primary node processor 26 to the backup processor 46
(step 265). The automatic redirection of requests to the backup processor is
also described in the U.S. patent application entitled "Network System with
Resilient Virtual Fault Tolerant Sessions" by Marc Desgrousilliers, commonly
assigned, filed concurrently with the present invention and incorporated by
reference herein.

DEPR:
Referring now to FIG. 4, which relates to SPX servers, typically the bulk of an application's time is spent not in establishing an endpoint or socket, but rather in sending and receiving data packets (step 290). After the connection has been established between client and server applications, the data that relates to the establishment of the connection is checkpointed (step 290). In the event of a takeover during the sending and receiving of data (step 295), if the network connection between the applications is in a connected state (step 297) and the connection between server and primary applications is "idle" (step 301), then the backup node may seamlessly and transparently assume the primary nodes responsibilities (step 305). Otherwise, if the applications are not connected or the connection is non-idle, the relevant sockets are reset (step 310). An idle connection is as defined above, and relates to the absence of a queue of data packets in the protocol process application.

CLPV:
checkpointing, when the protocol process application is not idle, transmitted packets from the primary node to the backup node.

CCXR:
714/13